Primljen: 23.11.2015.
Prihvaćen: 25.11.2015.

Stručni rad
UDK 004.03:004.62

Razvoj informacijskog sustava za evidentiranje podataka na primjeru obrasca zahtjeva za odgodu nastave i definiranje termina nadoknade implementiranog tehnologijom MVC tvrtke Microsoft

Development of the data storage information system for class postponement and substitution terms implemented with Microsoft MVC technology

[1]Saša Škvorc, [2]Mihael Kukec
[1]student Međimurskog veleučilišta u Čakovcu
[2]Međimursko veleučilište u Čakovcu
Bana Josipa Jelačića 22a, 40 000 Čakovec, Hrvatska
e-mail: [1] sskvorc@student.mev.hr; [2] mihael.kukec@mev.com

Sažetak: *Ovim radom opisuje se izrada prototipne web aplikacije korištenjem obrasca oblikovanja Model-Pogled-Kontroler (engl. Model-View-Controler – MVC) i tehnologije tvrtke Microsoft pod nazivom ASP.NET MVC 5. Cilj je prikazati i opisati mogućnosti ASP.NET MVC 5 tehnologije, kao i drugih korištenih alata i programskih okvira na primjeru izrade prototipne web aplikacije. Izrađena aplikacija predstavlja mogući pristup rješenju za računalno ispunjavanje i spremanje obrazaca koji se koriste na Međimurskom veleučilištu u Čakovcu i postoje u papirnatom obliku.*

Opisane su korištene metode i tehnologije te njihova upotreba i primjena u izradi web aplikacije iz praktičnog dijela rada. Prikazuje se primjena MVC obrasca oblikovanja koji je temelj korištenog programskog okvira u izradi aplikacije (ASP.NET MVC 5) te druge tehnologije i alati korišteni u izradi aplikacije. Navodi se tehnologija naziva „Entity Framework Code First" te je prikazan način njezine upotrebe na primjeru prototipne aplikacije.

Praktičnim dijelom rada uz upotrebu prethodno spomenutih metoda, tehnologija i alata izrađena je prototipna Web aplikacija koja se ne može smatrati gotovim, produkcijskim rješenjem za sustav upravljanja podacima i obrascima na Međimurskom veleučilištu u Čakovcu, već ona predstavlja ogledni primjer aplikacije koja može poslužiti kao temelj ili prvi prototip za analizu specifikacija i slučajeva primjene. Prototipnom aplikacijom je izrađena podrška za jedan obrazac, međutim kroz implementaciju za jedan obrazac stvoreni su uvjeti i

prikazan je način na koji je moguće ostvariti podršku za rad s novim obrascima. Aplikacija ima implementiran sustav za upravljanje korisničkim računima. Uz to, u aplikaciji je implementirano administratorsko korisničko sučelje za uređivanje podataka o studijskim programima, kolegijima, nastavnicima, dvoranama i korisnicima aplikacije što olakšava implementiranje dodatnih obrazaca koji se mogu naknadno dodati u aplikaciju prema potrebi te aplikacija može poslužiti kao stvarno rješenje za upravljanje obrascima u digitalnom obliku.

Ključne riječi: *web aplikacija, MVC, ASP.NET, Entity Framework*

Abstract: *This article describes the development of the prototype web application using MVC design pattern and Microsoft ASP.NET MVC 5 technology. The goal is to show and describe possibilities and features of ASP.NET MVC 5 technology as well as other tools and frameworks used in development of prototype web application. Developed application can represent possible approach and solution for building an information system for handling work with various application forms that are being used at Polytechnic of Međimurje in Čakovec.*

This article describes methods, tools and technology used in development of prototype application. Among others, article provides overview of MVC design pattern and Entity Framework Code First technology which was used for modeling and creating database in application development.

Implemented application is not the complete and final solution for management of application forms used at Polytechnic of Međimurje in Čakovec, but this prototype application can be used as a base for more complete solution. Developed web application implements support for one application form, and through that implementation it provides conditions and means to implement support for additional application forms in the future releases. Application has implemented user account management and administrative user interface for managing additional data which is being used in application forms. This enables easier upgrade of application with new features and expanding support for additional application forms.

Keywords: *web application, MVC, ASP.NET, Bootstrap, model, Entity Framework*

1. Uvod

U poslovanju raznih tvrtki, institucija, fakulteta pa tako i na Međimurskom veleučilištu u Čakovcu svakodnevno se koriste najrazličitiji obrasci u papirnatom obliku za prikupljanje podataka te artikuliranje i evidentiranje detalja o poslovnom procesu. Često se ti obrasci nalaze na računalu u digitalnom obliku, spremljeni u obliku datoteka nekog programa za obradu teksta. U konačnici, računalo se najčešće koristi za stvaranje dokumenta te ispis obrazaca na papir, a sam rad s obrascima obično je vezan uz obrazac na papiru koji se popunjava olovkom te se pohranjuje u registrator, fascikl ili ladicu. To otežava pristup, čuvanje i transparentnost rada s obrascima. Štoviše, postoje istraživanja koja pokazuju da korištenje e-maila u poslovanju povećava potrošnju papira za 40% (Sellen, Harper, 2003.).

2. Metode i tehnologije korištene u izradi aplikacije

Jedno od mogućih rješenja za olakšavanje rada s obrascima izrada je web aplikacije za upravljanje podacima koji bi se inače unosili na papirnate obrasce. Tehnologije koje su odabrane kako bi se to postiglo su ASP.NET tvrtke Microsoft uz obrazac oblikovanja model-pogled-upravitelj (engl. *Model-View-Controller* – MVC) verzije 5, sustav za upravljanje bazom podataka tvrtke Microsoft MS SQL, knjižnica programskog kôda za izradu elemenata sučelja naziva „Bootstrap“, okvir za objektno relacijsko mapiranje naziva „Entity Framework“ te programski jezici JavaScript i C#. U tekstu rada objasnit će se tehnologije, metode i alati koji bi se mogli koristiti u rješenju problema rada s obrascima na Međimurskom veleučilištu u Čakovcu. Valjanost opisanih postupaka provjerena je u praktičnom dijelu rada izradom prototipne web aplikacije kao primjer mogućeg pristupa rješenju rada s obrascima, kako bi se na Veleučilištu moglo učinkovitije upravljati obrascima. U ovom poglavlju slijedi opis najvažnijih metoda, tehnologija i alata koji su korišteni u izradi aplikacije iz praktičnog dijela rada.

2.1. Obrazac oblikovanja model-pogled-upravitelj

Model-pogled-upravitelj (engl. *Model-View-Controller - MVC*) obrazac je oblikovanja programskih rješenja koji se u računalnoj znanosti pojavljuje već dugi niz godina. Osmislio ga je Trygve Reenskaug 1979. godine prilikom posjete znanstvenicima Smalltalk grupe u poznatom centru Xerox Palo Alto Research Center (Reenskang, 2003.). Izvorni naziv „Thing-

Model-View-Editor“ kasnije je pojednostavljen i preimenovan u „Model-View-Controller“ ili skraćeno MVC. Primarna snaga i korist korištenja obrasca oblikovanja MVC je odvajanje pojedinih dijelova aplikacije u zasebne komponente ovisno o njihovoj namjeni (engl. *separation of concerns within an application*) (Booch et al.). U novije vrijeme obrazac oblikovanja programske podrške MVC pronašao je svoju primjenu u web aplikacijama te je izravno podržan u razvojnim alatima tvrtke Microsoft (Freeman, 2014.).

Izričito odvajanje pojedinih dijelova aplikacije u posebne cjeline dodaje određenu kompleksnost oblikovanju aplikacije, no niz različitih prednosti ipak nadmašuje dodatan trud koji se mora uložiti te opravdava primjenu obrasca oblikovanja MVC. Tri glavna dijela obrasca oblikovanja MVC jesu:

- model (engl. *Model – M*) – sadrži opis podataka s kojima se radi, kao i skup pravila prema kojima se podaci mijenjaju te provjerava njihova valjanost
- pogled (engl. *View – V*) – definira izgled korisničkog sučelja
- upravitelj (engl. *Controller – C*) – upravlja komunikacijom korisnika, ukupnim tokom izvršavanja aplikacije i logikom aplikacije (http://www.w3schools.com/aspnet/mvc_intro.asp.).

MVC obrazac oblikovanja od svog se predstavljanja koristio u više različitih programskih okvira (engl. *framework*) te ga se može primijeniti neovisno o programskom jeziku.

2.2. Entity Framework

Entity Framework (EF) je objektno/relacijski programski okvir (engl. *Object/Relational Mapping - O/RM*) te skup tehnologija koji podržava razvoj aplikacija koje su orijentirane na podatke. Pri tome predstavlja objektno-relacijski (engl. *object-relational*) alat za pridruživanje, odnosno mapiranje koji omogućava .NET programerima da rade s relacijskim podacima koristeći specifične objekte domene (Walther, 2009.). EF eliminira potrebu stvaranja većine programskog kôda za pristup podacima koje bez njegove primjene programer mora stvoriti.

Postoje dva općenita načina primjene EF-a. Prvi je kad već postoji baza podataka ili se ona oblikuje u početnom koraku, te se tek nakon toga stvaraju ostali dijelovi aplikacije. Drugi je kad se želi fokusirati na razrede domene i tek nakon toga stvoriti baza podataka prema razredima domene.

3. Ostvareno programsko rješenje

U praktičnom dijelu ovog rada izrađen je prototip aplikacije kroz čiju se izradu prezentiraju mogućnosti ASP.NET MVC 5 tehnologije, kao i drugih korištenih alata. Cilj je izrada sustava u obliku web aplikacije koji predstavlja mogući pristup rješenju za upravljanje obrascima koji se koriste na Veleučilištu.

3.1. Struktura direktorija aplikacije

Prilikom stvaranja ASP.NET MVC aplikacije razvojna okolina Microsoft Visual Studio automatski dodaje nekoliko datoteka i direktorija u projekt. ASP.NET MVC projekti stvoreni pomoću predloška „ASP.NET Web Application“ imaju osam glavnih direktorija. Tako se svi upravitelji smještaju u direktorij „Controllers“, modeli u direktorij „Models“, a pogledi u direktorij koji nosi ime pripadajućeg kontrolera, i to na način da taj direktorij predstavlja poddirektorij „View“ direktorija.

Tablica 1. Najvažniji direktoriji MVC aplikacije.

Direktorij	**Svrha**
/Controllers	Datoteke programskih razreda koji upravljaju URL zahtjevima
/Models	Razredi koji predstavljaju podatke i služe za rad s podacima
/Views	Predlošci za stvaranje korisničkog sučelja, tj. izlaznog HTML-a
/Scripts	JavaScript knjižnice i skripte (.js)
/fonts	Bootstrap fontovi
/Content	CSS, slike i ostali sadržaj
/App_Data	Podaci za operacije čitanja/pisanja (engl. *read/write*)
/App_Start	Konfiguracijski podaci za „Routing“, „bundling“ i Web API

Izvor: autori.

3.2. Izrada korisničkog sučelja

Oblikovanje korisničkog sučelja izrađeno je pomoću programskog okvira naziva „Bootstrap“ (Westhuizen, 2014.). Bootstrap uključuje gotove stilove kontrola korisničkog sučelja poput gumba, lista, ikona, fontova, izbornika i drugih, a definirani su u CSS

datotekama koje se nalaze u direktoriju naziva „Content“ (Tablica 1). Uz Bootstrap elemente korisničkog sučelja, koristi se i knjižnica programskog kôda naziva „jQuery“ (Joshi, 2013.) izrađena u programskom jeziku „JavaScript“ koja elementima korisničkog sučelja daje interaktivnost. Sve JavaScript datoteke nalaze se u direktoriju naziva „Scripts“.

Korisničko sučelje podijeljeno je na dva dijela. Prvi dio je predložak koji je zajednički i koristi se na svim stranicama, a u taj predložak ulaze zaglavlje koje sadržava izbornike te podnožje (engl. *footer*) stranice. Drugi dio je sam sadržaj stranice koji se dinamički stvara za svaku stranicu. Ti predlošci nalaze se u direktoriju naziva „Views/Shared“.

Izgled izbornika korisničkog sučelja ovisi o vrsti korisnika koji je prijavljen. Tako na primjer postoje različiti izbornici za neprijavljenog korisnika, prijavljenog korisnika i korisnika s administratorskim ovlastima. Kontrola, odnosno odlučivanje koji predložak će se učitati ovisi o vrsti prijavljenog korisnika, a programski kôd se nalazi u datoteci naziva „_ViewStart.cshtml“.

3.3. Usmjeravanje URL zahtjeva

Za pristup točno određenom resursu na mreži koriste se reference (adrese) koje jednoznačno određuju traženi resurs, to su tzv. URL adrese (engl. *Uniform Resource Locators – URL*) (Berners-Lee et al., 1994.).

Slika 1. Dio kôda „RouteConfig.cs“ datoteke.

```
public class RouteConfig {
       public static void RegisterRoutes(RouteCollection routes) {
              routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
              routes.MapRoute(
                     name: "Index",
                     url: "{controller}",
                     defaults: new { controller = "{controller}",
                            action = "Index" }
              );
              routes.MapRoute(
                     name: "Default",
                     url: "{controller}/{action}/{id}",
                     defaults: new { controller = "Home", action = "Prijava",
                     id = UrlParameter.Optional });
       }
}
```

Izvor: Programski kôd prototipne implementacije, autor.

Promet u ASP.NET MVC okviru započinje URL zahtjevom (engl. *requests*) te je tzv. ASP.NET okvir za usmjeravanje (engl. *Routing framework*) jezgra svakog ASP.NET MVC zahtjeva. ASP.NET usmjeravanje (engl. *routing*) je mapiranje prema obrascu. Prilikom

pokretanja, aplikacija registrira jedan ili više obrazaca usmjeravanja u tablici usmjeravanja i na taj način sustav usmjeravanja zna što učiniti sa zahtjevima koji odgovaraju tim obrascima. Kada sustav usmjeravanja (engl. *routing engine*) zaprimi zahtjev, uspoređuje URL zahtjev s URL obrascem u tablici. Ukoliko je pronađena odgovarajuća ruta, zahtjev se prosljeđuje odgovarajućem upravitelju i metodi.

Usmjeravanje (engl. *routing*) unutar ASP.NET MVC platforme služi za ispunjavanje dva glavna zadatka: (i) spajanje, odnosno mapiranje ulaznog zahtjeva (koji inače ne odgovara točno određenoj datoteci na disku, kao što je to često slučaj kod drugih programskih okvira) prema odgovarajućoj metodi kontrolera (engl. *controller action*) te (ii) konstruiranje izlaznog URL-a koji odgovara metodi kontrolera. Primjer prikazan na Slici 1 pokazuje obrazac prema kojemu se grade rute, a to je {controller}/{action}/{id}.

3.4. Zaštita podataka od neovlaštenog pristupa

Zaštita podataka od neovlaštenog pristupa mehanizmima autentifikacije i autorizacije jedna je od najvažnijih značajki koje web aplikacija ovog tipa mora zadovoljiti. Kako bi se ispunio uvjet sigurnosti, prilikom razvoja aplikacije posebnu pažnju treba posvetiti sigurnosti web aplikacije na način da se osigura od neovlaštenog pristupa i mogućih napada koji bi mogli ugroziti kako samu aplikaciju, tako i cijeli informacijski sustav u kojem se aplikacija nalazi.

3.4.1. Korisnički računi

Ograničavanje pristupa web aplikaciji iz praktičnog dijela ovog rada implementirano je korištenjem korisničkih računa. Pristup mogućnostima i korištenje same aplikacije dopušteno je samo korisnicima koji posjeduju korisnički račun. Za upravljanje korisničkim računima koristi se mehanizam „Identity“ dostupan u razvojnom okviru ASP.NET. Temeljen je na OWIN (Open Web Interface for .NET) knjižnici (http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity.). U izradi aplikacije koriste se individualni korisnički računi (engl. *Individual User Accounts*).

Kao baza za upravljanje korisničkim računima koristi se programski kôd u datoteci naziva „KorisnickiRacunController.cs“ s metodama „Login“ i „LogOff“. Uz datoteku „KorisnickiRacunController.cs“ aplikacija koristi i prethodno definiranu datoteku „IdentityModels.cs“. Unutar datoteke „IdentityModels.cs“ definiran je razred „ApplicationUser“ koja je naslijeđen iz razreda „IdentityUser“ te predstavlja entitet korisnika.

Razred „ApplicationUser“ modificiran je na način da je proširen potrebnim svojstvima za entitet korisnika prema potrebama web aplikacije iz praktičnog dijela završnog rada. Tako je razred ApplicationUser proširen poljima „Ime“, „Prezime“, „Administrator“, „Referada“ i „Prodekan“ što prikazuje Slika 2.

Slika 2. Dio kôda razreda „ApplicationUser“

```
public class ApplicationUser : IdentityUser {
       public async Task<ClaimsIdentity>
       GenerateUserIdentityAsync(UserManager<ApplicationUser>manager) {
              public string Ime { get; set; }
              public string Prezime { get; set; }
              public bool Administrator { get; set; }
              public bool Referada { get; set; }
              public bool Prodekan { get; set; }
              var userIdentity = await manager.CreateIdentityAsync(this,
                     DefaultAuthenticationTypes.ApplicationCookie);
              return userIdentity;
       }
}
```

Izvor: Programski kôd prototipne implementacije, autor.

Korisnički računi mogu se podijeliti u četiri grupe:

- Administrator – korisnik s administratorskim ovlastima koji ima pristup svim sekcijama aplikacije te može stvarati i modificirati ostale korisničke račune
- Prodekan – korisnik s posebnim pravima vezanim uz odobravanje ispunjenih obrazaca
- Referada – korisnik s posebnim pravima vezanim uz provjeru obrazaca koje su ispunili korisnici
- Korisnik – korisnik kojem je dopušten pristup samo određenim djelovima aplikacije.

Za razlikovanje vrste korisnika koristi se sustav korisničkih uloga (engl. *user roles*), a informacije su spremljene u pomoćnim tablicama u bazi podataka. U tu svrhu postoje četiri korisničke uloge. Prva je „Admin“ za korisničke račune s administratorskim ovlastima, a druga „NUser“ za korisnike. Uz ove korisničke uloge, u aplikaciji su dodane dvije dodatne uloge za korisnike s posebnim ovlaštenjima, a to su „Prodekan“ i „Referada“. Ova funkcionalnost izvedena je s relacijskim odnosima. Tako se u posebnoj tablici pomoću stranog ključa (engl. *foreign key*) uparuju korisnički račun i korisnička uloga.

Glavni korisnički račun s administratorskim pravima dodan je programski u metodi „Seed“ unutar datoteke naziva „Configuration.cs“. Datoteka Configuration.cs se nalazi u direktoriju naziva „Migrations“. Ova metoda izvršava se prilikom prvog pokretanja aplikacije i prilikom izvršavanja migracija uz uporabu naredbe „Update-Database“ (Slika 3). Uporaba naredbe „Update-Database“ opisana je u poglavlju „4.3.1 Postupak migracija“.

Slika 3. Kôd tzv. „Seed“ metode pomoću koje se dodaje glavni korisnički račun s administratorskim pravima.

```
protected override void Seed(HelloWorld.Models.ApplicationDbContext context)
{
// Dodavanje build-in administratorskog računa
context.Configuration.LazyLoadingEnabled = true;
var userStore = new UserStore<ApplicationUser>(context);
var userManager = new UserManager<ApplicationUser>(userStore);

context.Roles.AddOrUpdate(r => r.Name, new IdentityRole { Name = "Admin" });
context.Roles.AddOrUpdate(r => r.Name, new IdentityRole { Name = "NUser" });
context.Roles.AddOrUpdate(r => r.Name, new IdentityRole { Name = "Prodekan" });
context.Roles.AddOrUpdate(r => r.Name, new IdentityRole { Name = "Referada" });
context.SaveChanges();

if (!context.Users.Any(usr => usr.UserName == "administrator-obrasci@mev.hr"))
{
var user = new ApplicationUser { UserName = "administrator-obrasci@mev.hr",
Administrator = true, Email = "administrator-obrasci@mev.hr", Ime =
"Administrator", Prezime = "" };
userManager.Create(user, "Passw0rd!23");
context.Users.Add(user);
context.SaveChanges();

if (context.Users.Any(usr => usr.UserName == "administrator-obrasci@mev.hr"))
{
var result = userManager.AddToRole(user.Id, "Admin");
userManager.AddClaim(user.Id, new Claim(ClaimTypes.GivenName, user.Ime));
}
}
```

Izvor: Programski kôd prototipne implementacije, autor.

Pomoću glavnog korisničkog računa s korisničkim imenom "administrator-obrasci@mev.hr " kasnije se mogu dodati i administrirati ostali korisnički računi. Za upravljanje korisničkim računima koristi se posebni upravitelj „KorisniciController“ koji posjeduje metode za stvaranje, uređivanje i brisanje korisničkih računa (Tablica 2).

Tablica 2. Metode upravitelja Korisnici.

Metoda	**Opis**
Index()	Lista svih korisnika aplikacije
Detalji()	Pregled odabranog korisnika
Novi()	Stvaranje novog korisnika
Uredi()	Uređivanje odabranog korisnika
ResetLozinke()	Resetiranje lozinke odabranog korisnika
Obrisi()	Brisanje odabranog korisnika

Izvor: autor.

3.4.2. Ograničavanje pristupa

U cilju postizanja više razine sigurnosti i zaštite pojedinih dijelova aplikacije od neovlaštenog pristupa, u aplikaciji se koriste četiri vrste korisnika te se na temelju pripadnosti pojedinoj skupini dopušta, odnosno uskraćuje pristup pojedinim dijelovima aplikacije.
Ovdje treba razumjeti razliku između dva pojma, a to su:

- ovjeravanje autentičnosti ili autentifikacija (engl. *authentication*) – potvrđivanje identiteta korisnika, najčešće preko sustava prijave pomoću korisničkog imena i lozinke
- ovlaštenje ili autorizacija (engl. *authorization*) – provjeravanje ima li korisnik dopuštenje za pristup ili izvršavanje određene akcije.

Web aplikacija izrađena u praktičnom dijelu ovog rada ne dopušta pristup bez autentifikacije korisnika. To znači da samo prijavljeni korisnici s valjanim korisničkim računom imaju pristup aplikaciji.

Slika 4. Dio kôda upravitelja „Home“ koji dopušta pristup samo prijavljenim korisnicima.

```
[RequireHttps]
[Authorize]
public class HomeController : Controller {
       public ActionResultIndex() {
              return View();
       }

[AllowAnonymous]
public ActionResult Prijava()
{
       if ( User.Identity.IsAuthenticated )
       {
              return RedirectToAction("Index");
       }
       return View();
}

public ActionResult Kontakt()
{
       ViewBag.Message = "Kontaktirajte nas..";
       returnView();
}
```

Izvor: Programski kôd prototipne implementacije, autor.

Nakon prijave korisnika pristup pojedinim dijelovima aplikacije dopušta se ili uskraćuje pomoću autorizacije, odnosno provjerom ima li prijavljeni korisnik pravo pristupa

traženom dijelu aplikacije. Ukoliko korisnik nema pravo pristupa traženom resursu, on se automatski preusmjerava na stranicu za prijavu.

Ograničavanje pristupa pojedinim dijelovima aplikacije implementirano je pomoću atributa „Authorize“ koji su definirani u „AuthorizeAttribute“ razredu. Pomoću atributa „Authorize“ možemo odrediti, odnosno odobriti pristup točno određenim korisnicima ili tipovima korisnika. Koristeći korisničke uloge i svojstva korisnika možemo odrediti kojim je korisničkim ulogama ili korisnicima dopušten pristup. Primjerom na Slici 4. pokazuje se da iako je pristup svim metodama upravitelja „Home“ ograničen na razini upravitelja s atributom [Authorize], pristup metodi „Prijava()“ omogućen je pomoću atributa „[AllowAnonymous]“ i neprijavljenim korisnicima. Jednako tako, pristup upravitelju „Korisnici“ (i svim pripadajućim metodama) koji služi za upravljanje korisničkim računima dopušten je samo korisnicima koji imaju administratorske ovlasti.

3.4.3. Provjera unesenih podataka

Važna stavka sigurnosti aplikacije je i provjera podataka koje korisnici unose preko elemenata korisničkog sučelja za upis. Kako bi se osigurala točnost i valjanost podataka koje korisnici unose, sav se unos mora provjeravati te onemogućiti unos pogrešno formatiranih podataka. Za provjeru, tj. validaciju podataka koje korisnici unose u izrađenoj aplikaciji koriste se atributi naziva „Data annotations“. Oni se nalaze se u *System.ComponentModel.DataAnnotations* području imena, a služe za provjeru pojedinih polja u modelu podataka. Ti atributi definiraju uobičajene obrasce provjere poput provjere raspona vrijednosti polja i provjere obveznih polja (Tablica 3).

Tablica 3. Neki od korištenih atributa za provjeru u izrađenoj aplikaciji.

Atribut	**Značenje**
Required	Pokazuje da je svojstvo obavezno polje
DisplayName	Definira tekst koji će biti prikazan na formi i u validacijskim porukama
StringLength	Određuje maksimalnu dužinu polja s podatkom tipa string
Range	Određuje minimalnu i maksimalnu vrijednost numeričkog polja
DataType	Vrsta podataka

Izvor: autor.

Primjena atributa za provjeru valjanosti podataka prikazana je na Slici 5 na kojoj se nalazi primjer programskog kôda za provjeru valjanosti prilikom unosa lozinke.

Slika 5. Primjer kôda za provjeru valjanosti unosa lozinke.

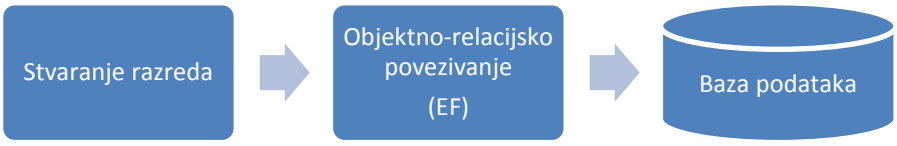
```
[Required(ErrorMessage = "Lozinka je obavezno polje")]
[StringLength(100,
      ErrorMessage = "{0} mora biti najmanje duljine {2} znaka.",
      MinimumLength = 6)]
[DataType(DataType.Password)]
[Display(Name = "Lozinka")]
public string Password { get; set; }
```

Izvor: Programski kôd prototipne implementacije, autor.

3.5. Stvaranje strukture i početno popunjavanje baze podataka

Sustav za upravljanje bazom podataka (SUBP) koji je korišten pri izradi web aplikacije je „SQL Server Express LocalDB“ tvrtke Microsoft (https://msdn.microsoft.com/en-us/library/hh510202.aspx.). Upravo je „LocalDB“ novorazvijena verzija „SQL Express“ baze podataka posebno stvorena za programere te je instalirana unutar razvojne okoline „Visual Studio“. To je pojednostavljena verzija SUBP-a te se pokreće na zahtjev i izvršava u korisničkom načinu rada (https://msdn.microsoft.com/en-us/library/hh510202.aspx.). Najčešće se verzija „LocalDB“ ne koristi za produkcijske web aplikacije, već samo tijekom razvoja aplikacije. Kasnije se baza podataka web aplikacije prelaskom u produkcijsku okolinu smješta na SUBT instaliran na samom poslužitelju.

Slika 6. Primjena Code First tehnologije.



Izvor: autor.

Za stvaranje baze podataka i svih tablica korišten je indirektni pristup s bazom podataka pri kojem se ne koristi jezik SQL. Takvim pristupom se u prvom koraku stvara programski kôd razreda (modela) iz kojih su kasnije stvoreni svi elementi za rad s bazom podataka. Naziva se „kôd prvo“ (engl. *code first* pristup) i podržan je tehnologijom EF

(https://msdn.microsoft.com/en-us/library/hh510202.aspx). Klasičnim pristupom se obično prvo stvara relacijska baza podataka korištenjem jezika kao što je SQL (engl. *strcutred query language*), a nakon toga se stvaraju razredi koji predstavljaju podatke zapisane u bazi podataka. Za razliku od klasičnog načina, uz pomoć „EF Code First“ tehnologije fokus se usmjeruje na oblikovanje domene i kreće se od stvaranja razreda koji predstavljaju entitete domene. Pri tome se koristi programski jezik za oblikovanje razreda te se pomoću razreda oblikovanih uz pomoć „EF Code First“ tehnologije stvaraju relacije u bazi podataka.

Pojednostavljeno, ovim pristupom najprije su stvoreni razredi modela koji predstavljaju entitete domene, nakon toga stvara se razred konteksta baze podataka (engl. *database context*), a zatim Code First API generira tablice u bazi podataka.

Najvažnija relacija u bazi podataka izrađene prototipne web aplikacije je relacija u kojoj se spremaju entiteti implementiranog obrasca „Zahtjev za odgodu nastave i definiranje termina nadoknade“. U bazi podataka postoje i druge relacije koje se koriste za upravljanje korisnicima aplikacije, kao i dodatne pomoćne relacije.

4. Primjer uporabe *entity frameworka* iz praktičnog dijela završnog rada

U ovom poglavlju slijedi opis postupaka korištenih prilikom izrade praktičnog dijela rada, tj. prototipne implementacije. Cilj je prikazati uporabu EF-a i „Code First“ tehnologije, odnosno pristupa koji je primijenjen prilikom razvoja aplikacije, a to je u prvom koraku stvaranje razreda modela te nakon toga stvaranje baze podataka uz pomoć „Code First“ tehnologije. Na primjeru dodavanja novog entiteta „Nastavnik“ opisat će se koraci i postupak koji je također primijenjen i u razvoju ostalih glavnih dijelova aplikacije.

4.1. Dodavanje modela

Prvi korak je stvaranje novog razreda s podatkovnim članovima koja opisuju svojstva entiteta nastavnik. Podatkovni članovi se u razred dodaju kao svojstva (engl. *property*) (https://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx). Koristeći razvojnu okolinu „Visual Studio“, unutar direktorija „Models“ stvorena je nova datoteka „Nastavnik.cs“ tako da se odabere direktorij „Models“, te u kontekstnom izborniku opcije „Add“, i zatim “Class“. Nakon stvaranja razreda Nastavnik potrebno je definirati sva polja razreda te osigurati provjeru unosa prilikom stvaranja novih objekata iz razreda Nastavnik. Na Slici 7. prikazan je programski kôd razreda.

Pomoću atributa „required“ (puni kôd: [Required(ErrorMessage="Ime je obavezno polje")]) polja razreda se mogu postaviti kao obvezna i odrediti tekst „pogreška“ koja će se prikazati korisniku aplikacije. Kao posljednji korak potrebno je u ApplicationDbContext razredu dodati kôd „public DbSet<Nastavnik> Nastavnici { get; set; }“. Ovaj kôd iz baze podataka vraća set n-torki iz relacije u kojoj su spremljeni entiteti tipa Nastavnik.

Slika 7. Kôd razreda Nastavnik.

```
public class Nastavnik {
       public int Id { get; set; }

       [Required(ErrorMessage="Ime je obavezno polje")]
       [Display(Name="Ime")]
       public string Ime { get; set; }

       [Required(ErrorMessage = "Prezime je obavezno polje")]
       [Display(Name = "Prezime")]
       public string Prezime { get; set; }

       [Required(ErrorMessage = "Email je obavezno polje")]
       [EmailAddress(ErrorMessage = "Krivi format e-mail adrese")]
       [Display(Name = "Email")]
       public string Email { get; set; }

       public string ImePrezime
       {
              get { return Ime + " " + Prezime;  }
       }
}
```

Izvor: Programski kôd prototipne implementacije, autor.

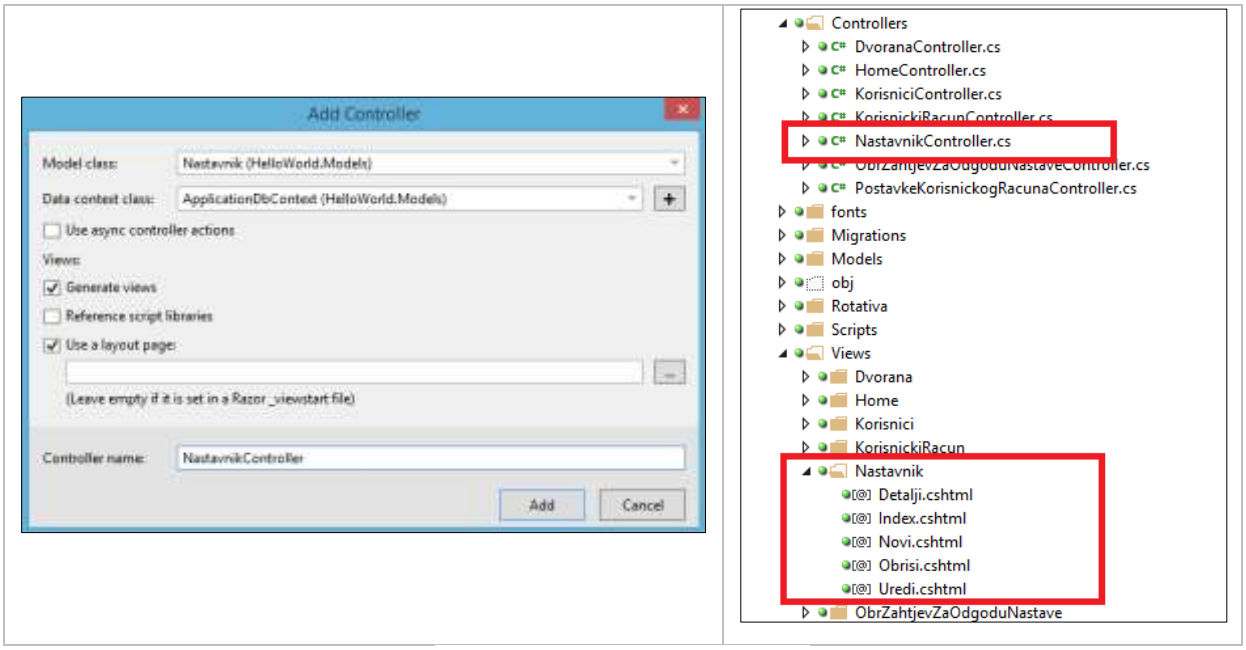
4.2. Stvaranje upravitelja za novi model

Nakon stvaranja razreda Nastavnik, u sljedećem koraku potrebno je dodati pripadajući upravitelj. Kao preduvjet ovom koraku nužno je kompilirati aplikaciju. Upravitelj se dodaje korištenjem funkcionalnosti „Scaffolding“ razvojnog okvira EF. To je alat za automatsko stvaranje programskog kôda za ASP.NET web aplikacije (http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview). Pomoću ovog alata na temelju stvorenog razreda modela automatski se stvara kôd za pripadajući upravitelj i kôd za pogled datoteke. Tako automatski stvoreni kôd predstavlja samo temelj s implementacijom najjednostavnijih mogućnosti te često nije dostatan u izvornom obliku pa ga je potrebno prilagoditi i proširiti.

Kako bi se pokrenula ova mogućnost, u sučelju razvojne okoline je potrebno odabrati direktorij „Controllers“ a zatim u kontekstnom izborniku opciju „Add“ i nakon toga „Controller“. Mogućnost automatskog stvaranja programskog kôda pokreće se odabirom opcije „MVC 5 Controller with views, using Entity Framework“. Odabir ove opcije omogućava da se prilikom generiranja upravitelja stvore i datoteke pogleda za osnovne

operacije stvaranja, čitanja, mijenjanja i brisanja podataka. U sljedećem koraku potrebno je odabrati model prema kojemu će se generirati novi upravitelj, kao i kontekst baze podataka te odrediti ime upravitelja što je pokazano na Slici 8., lijevo.

Slika 8. Odabir razreda modela, konteksta baze podataka i imena upravitelja (lijevo), stvorene i prilagođene datoteke (desno).



Izvor: autor, snimka zaslona.

Nakon što su određene sve postavke, tipkom „Add“ počinje automatsko stvaranje kôda. Kao rezultat, u direktoriju „Controllers“ stvorena je nova datoteka „NastavnikController.cs“ te je dodan direktorij „Views/Nastavnik“ i pogled datoteke za novi upravitelj što je prikazano na Slici 8, desno. Generirane datoteke upravitelja i pogleda sadrže sve potrebne metode za operacije s bazom podataka. Nazivi generiranih datoteka i metoda su na engleskom jeziku te ih prema potrebi možemo preimenovati i unutar njih implementirati vlastitu programsku logiku.

4.3. Sinkronizacija modela s bazom podataka

Nakon dodavanja novog modela „Nastavnik“ te pripadajućeg upravitelja i pogleda datoteka, model podataka više ne odgovara shemi baze podataka. Kako bi web aplikacija ispravno radila, potrebno je sinkronizirati nastale promjene u modelu podataka s postojećom shemom baze podataka. U tu svrhu koristimo alat naziva „Code first migrations“. To je alat

koji omogućava ažuriranje postojeće baze podataka s modelom podataka nakon promjena u razredima modela (https://msdn.microsoft.com/en-us/data/jj591621.aspx).

4.3.1. Postupak provođenja migracija

Usklađivanje programskog kôda, modela podataka s postojećim strukturama u bazi podataka, provodi se prema ovim koracima:

1. U razvojnoj okolini Visual Studio otvara se sučelje konzole „Package Manager Console“. Konzoli se može pristupiti tako da se u glavnom izborniku Visual Studio razvojne okoline odabere opcija „Tools“ a zatim „NuGet Package Manager“ te na kraju „Package Manager Console“.
2. Naredba „Enable-Migrations“ omogućuje migracije unutar projekta i dodaje direktorij „Migrations“ u projekt.
3. Naredba „Add-Migration <ime_migracije>“ dodaje migraciju sa zadanim imenom koja sadrži sve promjene u modelu koje su nastale nakon posljednje migracije.
4. Naredba „Update-Database“ će primijeniti sve potrebne migracije na postojeću bazu podataka.

Korištenjem ovog postupka struktura baze podataka bit će usklađena sa svim promjenama u razredima modela koje su nastale nakon početnog stvaranja baze podataka. Važno je istaknuti da je usklađenost strukture baze podataka i modela definiranih u razredima nužna te je nakon svake promjene u modelu potrebno izvršiti postupak usklađivanja, tj. migracije. Pri tome će sve datoteke vezane uz postupak usklađivanja biti pohranjene u direktorij naziva „Migrations“.

5. Zaključak

U izradi web aplikacije iz praktičnog dijela rada korištene su i proučene mogućnosti različitih tehnologija za razvoj web aplikacija. Iako je naglasak na korištenju Microsoft tehnologija poput ASP.NET MVC 5, EF i alata Visual Studio, u izradi web aplikacije korišteno je i mnogo drugih tehnologija otvorenog kôd a. Tako je, na primjer, za oblikovanje

korisničkog sučelja korišten Bootstrap programski okvir, a za kontrolu izvornog kôda Apache Subversion.

Aplikacija izrađena u praktičnom dijelu rada predstavlja moguće rješenje za centralizirano upravljanje obrascima koji se koriste na Međimurskom veleučilištu u Čakovcu. Iako ova aplikacija predstavlja samo prototip s implementiranim rješenjem za jedan obrazac, ona se može dodatno nadograditi tako da se dodaju i ostali obrasci. Aplikacija već nudi rješenje za administraciju korisnika aplikacije, kao i dodatne relacije u okviru baze podataka koje omogućuju lakši unos i uređivanje obrazaca.

Literatura

1. "ASP.NET MVC Tutorial." http://www.w3schools.com/aspnet/mvc_intro.asp. (01.01.2015.).
2. "ASP.NET Scaffolding in Visual Studio 2013." http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview. (22.06.2015.)
3. "Code First Migrations." https://msdn.microsoft.com/en-us/data/jj591621.aspx. (22.06.2015.)
4. "Getting Started with Entity Framework 6 Code First using MVC 5." https://msdn.microsoft.com/en-us/library/hh510202.aspx. (22.06.2015.)
5. "Introduction to ASP.NET Identity." http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity. (22.06.2015.)
6. "Properties (C# Programming Guide)." https://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx. (22.06.2015.)
7. "SQL Server 2016 Express LocalDB." https://msdn.microsoft.com/en-us/library/hh510202.aspx. (22.06.2015.)
8. Berners-Lee, T; Masinter, L.; McCahill, M. (1994). "RFC 1738 - Uniform Resource Locators."
9. Booch, G. et al. (2007). *Object-oriented Analysis and Design with Applications.* 3rd ed. Addison-Wesley Professional.
10. Freeman, A. *Pro ASP.NET MVC 5 Platform*, 1st ed. Berkely, Apress.
11. Joshi, B. (2013). *Beginning jQuery 2 for ASP.NET developers using jQuery 2 with ASP.NET web forms and ASP.NET MVC*. New York, Apress.
12. Reenskaug, T. (2003)."The Model-View-Controller (MVC) Its Past and Present," *Univ. Oslo Draft*, no. Mvc, pp. 1–16.
13. Sellen, A. J.; Harper, R.H.R. (2003).*The Myth of the Paperless Office*. Cambridge, MIT Press.
14. Walther, S. (2009). *ASP.NET MVC Framework Unleashed*, 1st ed. Carmel,SAMS.
15. Westhuizen, P. (2014). *Bootstrap for Asp.NET MVC incorporate Bootstrap into you ASP.NET MVC projects and make your website more user friendly and dynamic*. Birmingham, Packt Pub.